4.1 文件：

文件的本质是一个存放数据的容器。逻辑上，文件由一个以文件尾结束的连续字节流组成。物理上，文件由存储在磁盘这类稳定的介质中的一系列数据块组成。

文件由数据块和inode节点组成，数据块是实际存放数据的磁盘块，iNode节点里有文件的信息，如类型，属主，访问权限，时间戳。每一个inode有唯一的标识号，称为inode号，也称为文件顺序号。ls -s显示inode节点的大部分内容，ls -i显示节点号，stat显示inode的每一项内容

七种问价类型：普通文件，目录文件，块文件，字符文件，管道，符号链接，套接字。

register：这个关键字请求编译器尽可能的将变量存在CPU内部寄存器中，而不是通过内存寻址访问，以提高效率。

寄存器：数据从内存里拿出来先放到寄存器，然后CPU 再从寄存器里读取数据来处理，处理完后同样把数据通过寄存器存放到内存里，CPU 不直接和内存打交道。

4.1.1 stat(), fstat(),lstat函数

int stat(const char \*pathname, struct stat \*buf); int lstat(const char\* pathname, struct stat \*buf); int fstat(int filedes, struct stat \*buf); fstat返回的是与描述字相连的文件特性。当给定的文件不是符号链接的时候，stat和lstat函数一样，不然，stat返回链接所引用的文件特性，而lstat返回链接本身的特性，返回的文件特性存储在buf所指类型为stat的结构中

4.2 文件类型

文件类型由stat结构的成员st-mode给出

4.2.1：普通文件 简单的由一系列的字节组成，普通文件的格式要求是由访问它们的程序约定的，而不是操作系统强加的

4.2.2：目录文件 是一种特殊的文件，目录是由目录登记项组成的一张表，目录中的每一个文件和子目录在其中都有一个登记项，每一个目录登记项用来映射文件名至它对应的inode，它的格式很简单，仅有两个不同的域：inode号和引用这个inode的文件名

4.2.3 链接和link()函数

提供咯让多个文件名表示同一个文件的机制，好处是文件系统只需要存储文件的一个副本，每个文件的链接数记录在inode的st-nlink域。在相同或者不通同的目录中创建多个名称，每个均指向相同的inode节点，硬链接就是不同的文件名的inode号相同。文件的目录登记项也称为文件的硬链接。局限性：1：硬链接不能跨文件系统，2：除了超级用户之外，一般用户是不能创建目录的硬链接。注意：硬链接仅仅是目录中的一个登记项，创建一个硬链接只是在目录中增加一个登记项，他并不创建inode，也没有为他分配存放文件内容的磁盘空间 int link(const char \*exitingpath, const char \*newpath) 用newpath给出的名字创建一个新的目录登记项，该目录登记项引用现存文件exitingpath。

4..2.4：符号链接与symlink()和readlink()

符号链接是一种特殊的文件类型，其数据是另一个文件的名称

符号链接是指向另一个文件的特殊文件，符号链接不是直接指向文件的inode号，而是指向文件的路径名。符号链接有自己的inode，并且在磁盘上有一小片空间存放另一文件的路径名，如果是相对路径名，解释为相对于包含此符号链接的目录。符号链接比硬链接更为灵活，它可以跨文件系统，也可以与目录链接，甚至链接一个不存在的文件名

int symlink(const char \*path, const char \*sympath) 创建一个符号链接文件symlink，该文件指向path。符号链接没有实在的文件内容，它的内容只是另一个文件或者目录的路径名，系统打开文件的时候，仅从中读取路径，然后顺链接路径直到找到实际文件，在读取所指向的文件或者目录

因为open追寻符号链接，为咯读取符号链接文件本身 int readlink(const char \*pathname, char \*buf, int bufsize);读取符号链接所指文件的路径名于buf中，然后关闭该文件，注意：存在buf中的文件名字符串不是空字符终止的，readlink的正常返回值给出buf中实际存放的字符个数，错误返回-1

4.2.5：特别文件

特别文件也称为设备文件，不包含数据，是将物理设备映射到文件系统中的文件名。

特别文件用mknod()系统调用创建，并且与内核的一个软件相连，该软件称为设备驱动程序。当程序提出一个读写特别文件的请求时，这直接导致与它相连的设备动作，此动作负责在进程和设备驱动程序之间传递数据。特别文件一般位于/dev目录中

有两类特别文件：块特别文件和字符特别文件。块设备按固定大小且可随机访问的块来存储和执行I/O，块的大小通常是512字节或2的幂次方，例子有硬盘，软盘，CD\_ROM驱动器等，只有块设备才能包含文件系统。字符设备可以存储和传送任意大小的数据字符设备不能随机访问，因而不允许文件位定位操作，例子有终端，鼠标等。

用ls-l查看文件列表时，以字母b和c开头分别是块特别文件和字符特别文件

当通过块特别接口传送数据时，系统在内部的高速缓冲区内缓冲数据，并按一定的时间间隔将高速缓冲区中的数据写至设备。用户程序也可以通过sync()和fsync实现高速缓冲区和设备之间的同步。

通过字符特别接口传送的数据直接在设备驱动程序和控制进程之间传送，不需要经过系统的高速缓冲区。

特别文件在磁盘上没有数据块，但在它所贮存的文件系统中有永久的inode

函数mknod可以创建特别文件或着FIFO(管道)但只有超级用户才可以创建特别文件。

4.2.6：测试文件的类型

文件类型存储在stat结构的stat\_mode域中。St\_mode域包含两种信息：文件类型和文件方式位。可以用宏测试来判断文件类型，如先取出文件状态，然后用S\_ISREG(buf.st\_mode)等等。

4.3：文件的属主和用户组

每一个用户都有用户名和用户ID，并且相连的组和组ID。同样文件也有一个属主和属主所在的组，属主即是创建文件的用户，文件的属主可以对文件进行任何操作，属于同一个组的用户可以共享该组的文件。文件的属主用属主的用户ID来表示，，结构stat的成员st\_uid给出文件的属主ID,而st\_gid给出文件属主的组ID

实际用户ID，实际组ID指明实际用户是谁。有效用户ID，有效组ID，附加组ID用于文件访问权限测试。保存的调整用户ID，保存的调整组ID由exec保存。

文件首次被创建的时候，它的属主ID就是创建它的那个进程的有效用户ID，它的组ID就是那个进程的有效用户组ID或者就是进程创建它时所在目录的组ID

4.3.1：用函数改变文件的属主ID和组ID

int chown(const char \*pathname, uid\_t owner, gid\_t group); int lchown(const char \*pathname, uid\_t owner, gid\_t group); int fchown(int filedes, uid\_t owner, gid\_t group);

他们都是将指定的文件的属主id和组id改变为指定的大小。chown和lchown不同之处在于当pathname指定的文件时符号链接时，chown改变的是符号链接所指的实际文件，而lchown改变的是符号链接文件本身

普通用户不能将文件的用户ID改变为其他用户。虽然可以改变属于自己的文件的组ID，但仅仅能改变至自己所属的其他组

4.4：文件方式

文件访问权限由文件方式给出，文件方式确定文件可以由谁来读，写，执行。每一个文件均有一个文件方式与之相连，它由stat结构的st\_mode成员中的问价方式给出

4.4.1：文件访问权限

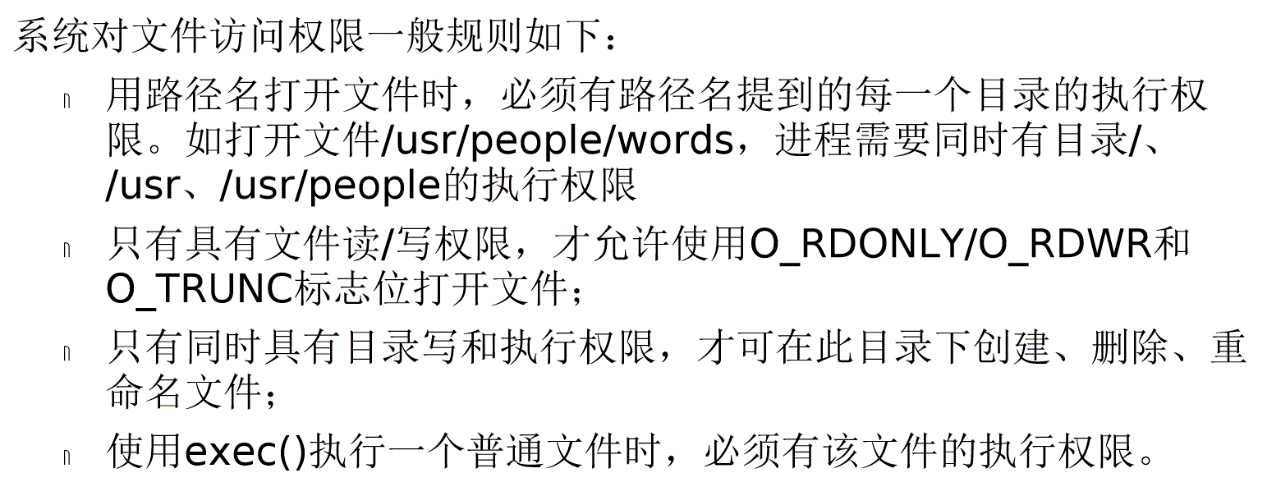
当用户企图访问文件的时候，它要么是该文件的属主，或者该文件属主同组的成员，要么是其他用户，每一种情形对应三种动作(读 写 执行)。文件方式一共有九位，他们按三位一组分类。通常9个方式位用3个八进制数字来表示，每一个八进制代表一类访问权限。如”751”按三位分解为”111 101 001”，表示文件属主有读写执行权限，文件的同组用户有读执行权限，而其他用户就只有读权限。每一位也对应一个宏常量

1：目录的访问权限解释

我们有某个目录的读权限，但没有该目录中文件的访问权限，我们仍然可以看到目录中文件的名称，访问权限，文件大小，访问时间，属主，组以及链接数等，但不能读目录中文件文件的内容。因为文件名存储在目录内，而不是文件内，目录的写权限允许创建，重新命名或删除目录内的文件，所有开放目录的写权限是比较危险的。目录的执行权限仅用于检查目录内的文件，因而称为检索权限。在一个目录和它的子目录内执行任何操作都需要有执行权限才行。没有执行权限那么就连访问此目录都不行



2：问价访问的一般规则



4.4.2：调整用户ID和调整组ID

当执行一个程序文件时，进程的有效用户ID通常就是实际用户ID，有效组ID就是实际组ID。文件方式字中的这两位分别叫做调整用户ID和调整组ID，他们的屏蔽码宏常量分别为S\_ISUID,S\_ISGID。例如：如果文件的属主是超级用户且该文件设置了调整用户ID位，当执行此文件时，无论实际用户是谁，执行它的进程都会具有超级用户特权。

用户执行权限位置上的字母s表示文件设置了调整用户ID位，它使得任何用户执行该文件程序时都可以往口令文件写新口令。

由于这种调整用户ID的程序将进程的有效用户ID改编为文件的实际用户，从而获得额外的特权，所以要小心使用。

4.4.3：sticky位

sticky位的屏蔽码宏常量是S\_ISVTX。

我有目录的写权限和目录中文件的读权限，我就能修改此文件。目录的写权限使我能够移动和重新命名该目录中的文件，以及在其中创建我自己的文件，文件的读权限使我能够复制该文件，当我有了副本后，我便可以对他进行任意修改然后我便可以用mv命令将修改后的副本覆盖原来的文件，从而达到修改文件的目的。但如果我将目录的属主设置sticky位后，则只有文件属主，目录属主和超级用户才可以删除，移动或者重新命名该目录中的文件。

4.5：确定和改变文件方式

4.5.1：umask函数

文件的访问方式除了由进程在调用open或creat创建文件时指定外，还受与进程相连的文件创建屏蔽的控制

如果umask的值为007，则表示屏蔽其他用户的所有访问权限，无论创建函数中指定mode参数是什么，新创建文件的其他用户访问权限位均为0.系统管理员在系统初启文件中用umask命令为每一个用户设置默认值。每一个进程从父进程继承它的umask值，shell进程的umask将传给它所执行的每一个进程。

如果我们只想让同组用户有读和执行权限，其他用户有执行权限，则可以 umask 026，即是”000 010 110”，一旦这个命令被执行，将来创建的任何文件都受它的保护

mode\_t umask(mode\_t cmask)；umask用cmask设置进的当前文件创建屏蔽，然后返回umask原来的屏蔽码，cmask参数由12个方式位中的任何一个按位或运算而形成。

改变进程的访问屏蔽值不影响其父进程的访问屏蔽值。

4.5.2：chmod和fchmod函数

Int chomd(const char \*filename, mode\_t mode), int fchomd(int filedes, mode\_t mode);设置文件的访问方式为mode，如果参数是符号链接，chomd改变的是它所指向的文件而不是符号链接文件本身，参数mode由12个方式位中的任何一个按位或运算而形成。

为咯改变一个文件的访问方式，进程的有效用户ID必须等于文件的用户ID，否则必须具有超级特权。

4.5.3：access函数

在编写调整用户ID的程序时，在读写一个文件之前必须明显地检查其用户是否原本就有对此文件的访问权限 int access(const char \*filename, int how); access判断调用进程是否允许按参数how指定的方式访问filename指定的文件，how的取值R\_OK,W\_OK,X\_OK,F\_OK(测试问价是否存在)；因为调整ID的程序并不改变实际ID，因而access函数根据进程的实际用户ID和实际组ID来判断。如果允许所有要求的访问，access返回0，否则返回-1.

用access查询的结果与open的检测结果不一致，执行该文件的用户和该文件的属主是不同的用户，虽然open能成功打开文件，但access函数明确告诉我们无权限访问这个文件，正是这种情况要使用access函数。

4.6：文件大小

stat结构成员st\_size以字节为单位给出文件的大小，这个域只对于普通文件和符号链接有意义。对于符号链接，文件大小是它所指文件的路径名含有的字符数。

Stat结构成员st\_blksize和st\_block分别给出文件的块大小和块数，UNIX系统中文件总是以块为单位存储在硬盘中，块的单位一般为512或者1024字节，st\_size是文件的实际大小，而块数和块的乘积是文件在磁盘中所占空间的大小。当一个文件的块的乘积远小于st\_size的时候，这个文件叫做稀疏文件。

UNIX文件系统为咯减少文件占用的磁盘空间，在磁盘中并不存储由NULL字符组成部分，而是记录两个非NULL字符之间NULL字符的个数，并在读该文件的时候才用NULL填充这个空间。

4.6.1：截断文件

对一个已经存在的普通文件，可以用函数截断，使其大小为指定的长度 int ftruncate(int filedes, off\_t length); int truncate(const char \*pathname, off\_t length);调用这函数必须得有该文件的写权限，参数length指定文件的字节大小，大于文件长度时，原文件结尾与新文件结尾之间将用NULL填充(也即在这之间产生了一个空洞)

4.7：文件时间

每一个文件都有三个时间戳与之相连：最近访问时间(最近一次读或执行该文件的时间)，最近修改时间(最近一次写该文件的时间)，最近特征修改时间(文件inode被改变的时间，如改变文件的访问权限，文件的链接数)。

一般而言，如果函数仅对文件本身进行操作，则更新文件的访问或者修改时间；如果函数涉及文件的inode，则更新文件的特征修改时间；如果操作涉及文件所在目录中的登记项，则还要更新这个目录的两个修改时间。

例如，当创建一个新文件的时候，此文件的三个时间均被设置为当前时间；此外由于在包含该文件的目录中国要增加一个新的登记项，所以还要更新这个目录的两个修改时间

4.7.1：utime和utimes函数

没有函数可以改变文件的特征修改时间，因为inode是由系统来维护的 int utime(const char \*pathname, const struct utimebuf \*times); utimebuf时专用于utime()的结构类型，它的两个成员均为日历时间，分别是文件的访问时间和文件的修改时间。参数times是空指针的时候设置为当前时间，此时进程的有效用户ID必须等于文件的用户ID或者就是要有文件的写权限。如果不是空指针，设置为指定时间，此时，要么进程的有效用户ID必须等于文件的用户ID，要么必须是超级进程，仅仅有文件的写权限是不足够的。Int utimes(const char \*path, const struct timeval values[2]);第二个参数是指向timeval结构的数组，其中values[0]指定文件的访问时间，values[1]指定文件的修改时间，timeval结构以微妙的精度指定时间

4.8：文件的删除与换名

不在需要的文件和目录应当删除，以免占用磁盘空间。由文件的硬链接我们知道，删除文件的动作实质上是删除文件名，如果此名字是文件的唯一名字，则文件也同时被删除，如果还有其他名字，则它仍然还可以被其他文件名访问，unlink和rmdir分别用于删除一个文件和一个目录，此外c库还提供了一个更为方便的函数，remove删除文件名和目录。文件换名则通过函数rename来实现。

4.8.1：删除文件和目录

int unlink(const char \*pathname);删除由pathname给出的文件名，并减少此文件的链接计数，如果该文件还有其他链接，文件将保持存在，如果是唯一文件名，则文件本身被同时删除。当最后的链接被删除的时候，如果有其他进程打开文件，链接将在unlink返回之前删除，但文件的内容要等到所有对该文件的访问关闭之后才删除。实际上，每当文件被关闭，内核要检查文件的链接数，当为0的时候，文件的内容便被删除。注意：pathname不能是目录，除非是超级进程。调用进程必须有指定文件的写权限和执行权限，此外，如果目录设置了sticky位，进程还必须是文件或目录的属主，或具有超级特权。

unlink删除一个文件后，只要没有关闭该文件，进程仍然可以读写它，unlink这种特性常常用来保证一个临时文件不会因为疏忽而留在系统中，在open或者creat一个临时文件后，立马就删除它，但之后没有关闭它之前仍然可以对它进行操作。

int rmdir(const char \*pathname)；此目录应该是一个空目录，与unlink不同，即使有进程仍然打开此目录，它也不能对该目录有所动作，因为此目录即将被删除。函数remove在删除文件上等同于unlink，在删除目录上等同于rmdir, int remove(const char \*pathname)

4.8.2：文件换名

Rename用于更改文件或者目录的名字，int rename(const char \*oldname, const char \*newname);oldname给出要更改的文件的路径名，参数newname指向文件的新路径名。注意：1：更改文件或者目录时，涉及所在目录的登记项，所以调用进程必须有这两个目录的写权限。2：如果newname是一个已经存在了的文件名，系统将删除它的链接，然后用newname替换oldname。3：如果old是一个目录名，更改的是目录名，此时newname或者不存在，或者必须是一个空目录，如果newname是一个已存在的空目录，则删除此目录并把oldname改名为newname，此外newname不得包含命名oldname的路劲前缀。4：如果oldname指向一个命名符号链接的路径名，则该符号链接被改名；如果newname指向一个符号链接的路径名，则它原来的链接被删除。

4.9：目录操作

char \*getwd(cahr \*pathbuf); char \*getcwd(char \*pathbuf, size\_t size); int chdir(const char \*pathname); int fchdir(int filedes);

函数getwd将工作目录的绝对路径名复制到进程提供的字符数组中，然后返回指向该数组的指针，getcwd需要提供大小。函数chdir使得pathname指定的目录成为调用进程的当前工作目录，调用chdir的进程必须得有pathname所有路径分量的执行权限，并且指定的长度不能超过PATH\_MAX，chdir改变的只是调用进程的工作目录，它对父进程的工作目录没有影响，当给出的是符号链接时，chdir顺符号链接定位至所引用的目录

4.9.2：创建目录

函数mkdir用于创建目录 int mkdir(const char \*pathname, mode\_t mode);该函数创建一个名为pathanme的空目录，参数mode指定目录的访问方式位，这些位受到文件创建屏蔽码的修正，这个新建目录的用户ID，及它的拥有着，被设置为调用进程的有效用户ID，它的组ID将继承其父进程的组ID或者进程的有效组ID。若调用成功，mkdir将会更新该目录的st\_atime,st\_ctime,st\_mtime，同时更新其父目录的st\_ctime,st\_mtime，然后返回0；

由pathname指定的新目录的父目录必须存在，且调用进程必须有该父目录的写权限及pathname设计的各个分路径目录的搜寻权限

4.9.3：读目录流

目录也是文件，但目录中的数据不是由字节组成的，而是由目录项组成的，因此对目录执行操作的函数使用系统定义的一种结构类型为DIR的结构，这种结构称为目录流